AMENDMENTS TO THE SPECIFICATION

Please replace the existing title of the application with "Performing SQL Query Optimization by Simplifying Sub-Expressions."

Please amend paragraph [0003] of the specification as shown below:

[0003] Implementations of the invention may include one or more of the following. Each expression may include one or more sub-expressions. Expression optimization may include, for each expression that has a form selected from the group of "SE+0," "SE*1," and "SE/1," where SE is a sub-expression, reducing the expression to SE. Expression optimization may include, for each expression that has a form selected from the group of "SE*0," "0/SE," and "0 MOD SE," where SE is a non-nullable sub-expression, reducing the expression to 0. Expression optimization may include, for each expression that has a form F(C), where F is a function and C is a constant and F(C) returns the return value, reducing the expression to a return value. The method may include, for each sub-expression that includes a sub-expression, simplifying the sub-expression. SE may be nullable if it includes a nullable column. SE may be nullable if it belongs to an inner table of an outer join. The query may be represented by a tree that includes one or more nodes. The query may include an assignment list clause that includes one or more of the expressions. The query may include a WHERE clause that includes one or more of the expressions. Further query optimization may include determining a satisfiability of the database query. . Further query optimization may include determining a transitive closure of the database query. . Further query optimization may include determining one or more plans for executing the query. One or more of the plans may include scanning a table to locate rows that satisfy one or more conditions and summing one or more columns in the rows that satisfy the one or more conditions. Further query optimization may include two or more optimizations selected from: determining a satisfiability of the database query; determining a transitive closure of the database query; determining one or more plans for executing the query; and selecting an optimal plan ~~from~~ **for** executing the database query.

Please amend paragraph [0013] of the specification as shown below:

[0013] The techniques for processing database queries disclosed herein have particular application, but are not limited, to large databases that might contain many millions or billions of records managed by a database system ("DBS") 100, such as a ~~Teradata Active Data~~

~~Warehousing System~~ **TERADATA ACTIVE DATA WAREHOUSING SYSTEM** available from ~~NCR Corporation~~ **TERADATA CORPORATION**. Figure 1 shows a sample architecture for one node 105_1 of the DBS 100. The DBS node 105_1 includes one or more processing modules $110_{1...N}$, connected by a network 115, that manage the storage and retrieval of data in data-storage facilities $120_{1...N}$. Each of the processing modules $110_{1...N}$ may be one or more physical processors or each may be a virtual processor, with one or more virtual processors running on one or more physical processors.

Please amend paragraphs [0026] and [0027] as shown below:

[0026] Fig. 6 shows an example system for scanning an addition, subtraction, multiplication, division, or modulo expression trees (block 515). The system calls the scan function (block 320) and passes the left tree (*e.g.*, the tree whose root is the left child of the current node, including the left child's descendants) and the clause type to the scan function. Then, the system calls the scan function (block 320) and passes the right tree (*e.g.*, the tree whose root is the right child of the current node, including the right child's descendants) and the clause type to the scan function. Then, if the clause type is an assignment list (block 605) and the expression is not within an alias list (block 615), the system returns the tree (block 620). ~~On~~ **One** example system determines whether it is in an assignment list by incrementing a "WithinConv" count by one each time the system enters an alias clause, and decrementing the "WithinConv" count each time the system ~~exists~~ **exits** an alias clause. Otherwise the system calls a prune function (block 610, which is shown in greater detail in Figs. 11A and 11B) and passes the function the tree and the "CurrentRet" variable (discussed below with respect to Fig. 5). In certain implementations the tree and other variables are passed by reference.

[0027] Fig. 7 ~~show~~ **shows** an example system for scanning a SELECT clause tree (block 525). The system begins by pushing the tree onto the stack referenced by the variable name "CurrentRet" (block 705). This variable will give other portions of the system access to the entire SELECT clause tree**[[,]]** when they are evaluating an expression or a sub-expression within the SELECT clause tree. The system then loops once for each clause in the statement (blocks 710 and 715). Within the loop, the system calls the scan function (block 320) and passes the tree corresponding to the clause and the clause type to the scan function. After exiting the

loop, the system pops the tree referenced by the variable "CurrentRet" off of the stack. The system then returns the tree (block 725).

Please amend paragraph [0031] as shown below:

[0031] If the function call is represented in the tree in the manner described above, the system calls the scan function (block 320) and passes the function name tree, (*e.g.,* the function name node and ~~it's~~ **its** descendants) and the clause type to the scan function. The system then calls the scan function (block 320) again and passes the parameter list tree, (*e.g.,* the parameter list node and ~~it's~~ **its** descendants) and the clause type to the scan function. The system then determines if the parameter list contains only constants (block 1005) and, if so, the system evaluates the function call for the values in the parameter list (block 1010) and replaces the function call tree with the result returned from the function call (block 1015). After block 1015, or if the parameter list does not contain only constants, the system returns the tree (block 1020).

Please amend paragraphs [0034] and [0035] as shown below:

[0034] Note that some of the example ~~simplification~~ **simplifications** listed above are possible only if the column or variable cannot take a null. In one example system, the nullability of a variable or column in a database is determined by the definition of the column. For example, table ta1, defined above contains the following column definitions: "x1 INT NOT NULL, y1 INT, z1 INT." Based on these definitions column x1 is not nullable, while columns y1 and z1 are nullable.

[0035] An example system for pruning expressions (block 610) is shown in figures 11A and 11B. In Fig. 11A the system receives a tree and a "CurrentRet" variable (block 1102). The system copies the left expression (*e.g.,* the tree formed by the left child of the root node and ~~it's~~ **its** descendants) to L (block 1104) and the right expression (*e.g.,* the tree formed by the right child of the root node and ~~it's~~ **its** descendants) to R (block 1106). The system determines if the tree is a multiply tree (*e.g.,* the root node of the tree is a multiply node) (block 1108) and, if so, the system evaluates the multiply expression (block 1110, which is shown in greater detail in Fig. 12).

Please amend paragraph [0041] as shown below:

[0041] In block 1142, the system determines if the tree is a subtraction tree (*e.g.*, the root node of the tree is ~~an~~ **a** subtraction node) (block 1132), and if so, tree the system proceeds to block 1142, otherwise the system returns the tree (block 1130). If the tree is a subtract tree, the system determines if R is equal to "0" (block 1144), and if so, the system returns L (*e.g.*, X - 0 = X) (block 1146), otherwise the system returns the tree (block 1130).